The zx2c4 `pass` password manager

BV

[2015-07-08 Wed 19:57]

I'm moving from KeePassX and LastPass to zx2c4 `pass` command line password manager.

1 Getting started

1.1 Installation

Pass is available as a Debian/Ubuntu package or easy to install from source.

1.2 Initialization

Initialize the store using an encryption subkey created just for this purpose.

```
$ pass init '0x4B774E09F6E1EDE9'
mkdir: created directory '/home/bv/.password-store/'
Password store initialized for 0x4B774E09F6E1EDE9
$ pass git init
```

2 Import

2.1 KeePassX

Use the KeePassX GUI to export to XML.

```
$ git clone http://git.zx2c4.com/password-store
$ cd password-store/contrib/importers
$ python keepassx2pass.py ~/keepassx.xml
Importing ....
$ rm ~/keepassx.xml
```

2.2 LastPass

Use LastPass web UI to download CSV.

```
$ ruby lastpass2pass.rb ~/lastpass.csv
$ rm ~/lastpass.csv
...
46 records successfully imported!
```

3 Integration

3.1 Bash

The main integration with the command line is through completion. When installed as a system package doing the usual:

```
source /etc/bash_completion
```

should be what is needed. If `pass` is installed from source it may be required to copy the `pass.bash-completion` script somewhere and source it explicitly.

3.2 Emacs

Install `password-store` from `package-list-packages` then use `password-store-*` functions.

3.3 X11

Pass is command-line oriented but can be easily and well integrated into X11 desktop environments.

3.3.1 Selection

By default `pass` uses `xclip` to send the password to the X11 "clipboard". This is not the selection buffer which gets pasted with Mouse-Button-3. To fix:

```
$ export PASSWORD_STORE_X_SELECTION=primary
```

3.3.2 dmenu

Dmenu is a lightweight mechanism to display a bar across the top/bottom of your X11 screen in order to make a selection. Zenity could also be used but dmenu is fast, noninstrusive and requires no mousing. Pass comes with passmenu to integrate pass with dmenu. I've hacked my copy to:

- set the store selection to `primary` (if used)
- make sure my copy of `pass` is in the `PATH`
- set the font to use for `dmenu`

Test with

```
$ passmenu --type
```

3.3.3 Sawfish

My X11 sessions are based on the Sawfish WM + MATE DE. To integrate Pass I bind the above command to the `Super-s` key. I also make a little `runmenu` script which essentially just calls `dmenu_run` and bind that to the `Super-r` key.

To use a password in this environment, I put focus to the targeted password field, type `Super-s`, select the password key using `dmenu` functionality, hit `ENTER` and provide my GPG passphrase if needed. The password is then auto-typed.

4 Git tricks

Pass can use `git` to manage the `~/.password-store`. Here are some tips/tricks.

4.1 Sync to remote

Following this mailing list post set up for syncing to a remote git repository. The `local` machine is where the initial Pass password store exists.

```
remote$ git init --bare ~/git/password-store

local$ pass git remote add origin ssh://USE@REMOTE:/home/USER/git/password-store
local$ cd ~/.password-store
```

```
local$ git push --set-upstream origin master
local$ cat <<EOF > .git/hooks/post-commit
git pull --rebase
git push
EOF
local$ chmod +x .git/hooks/post-commit
```

Test with:

```
local$ git generate foo 20
[logging from git showing a pull/push]
```

Note, only do this on password stores where generating the ssh/git traffic on each change to the store is acceptable.

4.2 Second local

Other local password stores using the remote are set up similarly.

```
$ pass init '0x4B774E09F6E1EDE9'
$ pass git init
$ pass git remote add origin ssh://bv@haiku:/home/bv/git/password-store
$ pass git branch --set-upstream-to=origin/master master
$ pass git pull --rebase
$ pass show foo
```

5 Usage

5.1 Organization

After importing from KeePassX and LastPass I had a hodge-podge of directories and files. With KeePassX I had folder organization that roughly followed a domain- or task-based hierarchy. LastPass tends to impose its own organization. KeePassX helps to enforce some organization by encouraging making new entries in the context of old ones while LastPass follows more a capture-and-store approach and leaves any context forming as a follow-on activity (which I never did). Finally, the conversion process uses name or title of the entry with some character munging to make it file-system-friendly. To make matters worse, both my KeePassX and LastPass stores have a lot of overlap (and yet worse, one whole folder came from a previous use of Revelation). All in all, it's a mess and the only solution is develop some sorter or just deal with things manually.

The organization I chose is a reverse domain name hierarchy + group/application + user but flattened. For example, the password for a my account in the EDG wiki goes at:

```
gov/bnl/phy/www/edg/wiki/bv
```

Some things like SSH key passphrases or domain passwords transcend some levels of the hierarchy so I have:

```
ssh/<some-name>
gov/bnl/domain/bviren
gov/bnl/phy/domain/admin
```

5.2 Conflict with Gnome Keyring Manager

```
$ pass show dir/name
gpg: WARNING: The GNOME keyring manager hijacked the GnuPG agent.
gpg: WARNING: GnuPG will not work properly - please configure that tool to not interfe
...
```

Oh my. Oh, hell, why not.

```
$ sudo sed -i s/AGENT_ID/AGENX_ID/ `which gpg2`
```

6 Links

- Hacker News entry on Pass with some good ideas

7 Ideas

One thing that makes `pass` great is that it's simple and open to interpretation. Here are some ideas for further development.

7.1 Hashed key index

By default, `pass` shows the key by which a password is looked up in plain text as the directory/file name. It is typical to usage domain of the password into this key which means anyone able to see the password store can know what domains one accesses. A common method suggested to combat this is simply making the password store reside in an ENCFS or other type of encrypted container.

Another option is to use a (SHA1) hash for the file name and maintain an encrypted index file that associates a human-readable key with that hash. This key can then be any sort of meta data.

```
# ~/.password-store/.pass-index
da39a3ee5e6b4b0d3255bfef95601890afd80709 file://some/entry/filename
da39a3ee5e6b4b0d3255bfef95601890afd80709 file://someother/entry/filename
da39a3ee5e6b4b0d3255bfef95601890afd80709 mailto:user@example.com
da39a3ee5e6b4b0d3255bfef95601890afd80709 http://example.com/app
```

The example shows using URL schema to name the key (with `file://` being left implicit)

For this to be useful, `pass`'s completion functions would need support added. To help write them more succinctly and consistently between shells, the `pass` command itself should provide some abstraction:

```
$ pass complete --scheme file some
some/entry/filename
someother/entry/filename
$ pass complete --scheme mail example
user@example.com
```

Some problems with this idea mostly have to do with complexity and incompatibility.

There is a slippery slope with this index as it's not much more of a leap to specify using a "proper", single-file database. But, this one step still preserves the one-password-per-file structure which makes things amenable to using git.

Some conversion process is needed that will take a nominal password store and make the index, renaming/moving/removing the plain-named entries.

If the hash is based on the content of the entry, then any edits needs to rewrite the hash for the filename as well as the index. Git is already doing this so maybe it's better implemented on top of a bare git repository (but see the slippery slope above).

Maybe a better, simpler, approach is to merely augment the `pass` status quo and use a file naming scheme that merely obfuscates the entry's domain. Then, the index can become a simple key-filename lookup to allow for more memorable keys.

```
# ~/.password-store/.pass-index
some/entry/filename.gpg alias: an-alias
```

```
some/entry/filename.gpg alias: an alias with spaces
some/entry/filename.gpg email: user@example.com
some/entry/filename.gpg url: http://example.com/app
```

This index can processed regenerated periodically to remove files that no longer exist, warn about unindexed files. It could also be fully generated based on interpreting the entries.

7.2 Template-based Filters

At its core, the only semantic interpretation `pass` makes on an entry is that the first line is the password. A common extension, and one that is supported by at least the two converters I applied, is that subsequent lines contain named fields and/or free-form text.

The LastPass converter produces entries like:

```
<password>
---
<category> / <site> / <hierarchy>
username: <username>
password: <password>
url: <url>
....
```

The KeePassX produces:

```
<password>
username: <username>
url: <url>
comment: <free-form, multi-line text>
```

The idea with template-based filters is to develop a parser of the commonly created entries, make the fields available to a template system to produce new output forms. For example,

```
$ pass show <name> | pass-filter auto-type-user-pass.j2
```

This might be then used to provide login auto typing like:

```
xdotool - <<<"type --clearmodifiers -- $(pass show <name> | pass-filter auto-type-user
```

Where the (Jinja2) template file might look like:

```
{{ username }}\t{{ password }}\n
```

The `pass-filter` command would be responsible for trying to guess the entry format or be told it explicitly

```
$ pass show from-keepassx/entry | pass-filter
$ pass show from-lastpass/entry | pass-filter --lastpass
$ pass show json/entry_in_json | pass-filter --json
$ pass show yaml/entry_in_yaml | pass-filter --json
```